Vol.2012-HPC-135 No.20
2012/8/2

クロスコンパイル環境でインストール時チューニングを容易にするミドルウェアの開発

鴨志田 良和[1,a)]

概要：本稿はクロスコンパイルされた実行ファイルとそうでない実行ファイルを判定する機構と，execve システムコールをフックし，クロスコンパイルされた実行ファイルを実行しようとすると，自動的にリモートホストへ実行を転送する機構を有する TLDT というツールを提案する．TLDT を利用することにより，ローカルホストから通常の実行ファイルとリモートホスト向けの実行ファイルが透過的に実行可能になるため，ATLAS を始めとするインストール時自動チューニングを行うソフトウェアなど，クロスコンパイル環境でのビルドが困難なソフトウェアを容易にビルドできるようになる．

Design and Implementation of Tool Simplifying Install-time Auto-Tuning for Cross-Compilation Environments

KAMOSHIDA Yoshikazu[1,a)]

Abstract: This paper proposes a tool called the TLDT (Transparent Load Distribution Tool), which runs two types of executables, one for a local host and one for a remote host on a cross-compilation environment, transparently. Transparent execution is implemented by interception of an execve system call and automatic detection of the differences between both types of executables. Using TLDT enables us to build various software applications which are difficult to build on a cross-compilation environment, including software with an install-time auto-tuning facility like ATLAS.

1. Introduction

近年のスーパーコンピュータは種々のコンピュータを組み合わせたシステムとして構成されている．中でも，実際の高性能計算を行う「計算ノード」と，プログラムのコンパイルやインストールを行う「ログインノード」またはフロントエンドノードは，あらゆるシステムに不可欠なものである．一部のスーパーコンピュータには，計算ノードとログインノードに異なるインストラクションセットアーキテクチャを採用し，計算ノードの性能向上とログインノードの利便性を両立しようとしているものがある．そのようなシステムには，プログラムをビルドするためにクロスコンパイラが提供されている．しかしながら，既存のソフトウェアが必ずしもクロスコンパイルをサポートしているわけではないため，クロスコンパイル環境でのビルドがうまくいかないことがある．典型的な例が，ビルド時に，まずソースコード生成するプログラムをコンパイルし，それを実行して，最終的に必要とされる実行ファイルのためのソースコードを生成するような処理を含む場合である．クロスコンパイル環境においてはソースコード生成プログラムは，実行環境 (計算ノード) で実行されるバイナリとなるが，クロスコンパイル環境でのビルドをサポートしていないソフトウェアの場合，それをビルド環境 (ログインノード) で実行しようとしてしまい，エラーとなってしまう．Makefile 等を書き換えて対応することも可能だが，大きなソフトウェアでは変更箇所が多岐にわたることもありうるため，透過的な解決策が必要である．インストール時自動チューニングを行うソフトウェアは，そのような，クロスコンパイルをサポートすることが困難なソフトウェアの一

[1] 東京大学
The University of Tokyo, 2-11-16 Yayoi, Bunkyo-ku, Tokyo 113–8658, Japan
[a)] kamo@cc.u-tokyo.ac.jp

ⓒ 2012 Information Processing Society of Japan

種である．

自動チューニングは，実行時，実行環境，あるいは入力データ等から取得できる様々な情報を利用して，プログラムから，コンパイラの静的な解析よりも優れた性能を引き出す技術である．様々な自動チューニング技術の中でも，インストール時チューニングは，プログラムのインストール時に判明している情報を利用するもので，チューニング可能なパラメータの最適な組み合わせを探索するコストの高い処理を，ソフトウェアのインストール時だけに行えばよいため，達成可能な性能向上と利用者の利便性のバランスを考えると大変重要な技術である．

インストール時チューニングの典型的な処理の流れは，以下のとおりである．

(1) 実行環境の情報を得るためのプログラムをコンパイルする
(2) チューニング可能なパラメータの組み合わせをひとつ選ぶ
(3) 選んだパラメータを使用して 1 のプログラムを実行する
(4) 1 から 3 を，すべてのパラメータの組み合わせについて繰り返す
(5) 最良の性能となるパラメータの組み合わせを選択する

ここで，チューニング可能なパラメータの種類は数多くあり，種類ごとに様々な値を試行する必要があるため，探索すべき範囲は相当に広い可能性がある．

この処理の流れの中で，3 のプログラム実行の部分は，計算ノードで行う必要があり，クロスコンパイル環境でこのステップを正しく実行するためには，適切にバッチジョブを投入し，計算ノードで処理を実行するようにしなければならない．バッチジョブの実行は非同期的であるため，計算ノードとログインノードのアーキテクチャが同一である場合には単純にコマンドを実行すればよかったところを，ジョブ投入，実行完了待ち，終了ステータスの処理を行うように変更する必要がある．Grid Engine の `qrsh` のように，`rsh` に似たインタフェースを持つバッチジョブ管理システムであれば，比較的簡単に計算ノードでの処理に切り替えることが可能な場合もあるが，ソフトウェアが大規模で複雑になると，必要なすべての箇所に上記の変更を加えることはより困難になる．

このような問題を透過的に解決するため，我々は TLDT というツールを提案する．TLDT は計算ノード向けの実行ファイルを自動的に判別し，ログインノードでそのような実行ファイルを実行しようとした場合に，自動的に計算ノードでの実行に切り替える．この処理は透過的に行われるため，利用者から見ると，計算ノード用のバイナリがあたかもログインノードで実行されるように見えることになる．このツールは，クロスコンパイル環境で様々なソフトウェアをインストールする際の生産性を大きく向上させることができると我々は考えている．

現在運用されているスーパーコンピュータの一部には，ログインノードとは別に，計算ノードと同じアーキテクチャを持ち，かつ対話的な処理が可能なインタラクティブ環境を提供しているものもある．多くの場合，ログインノードのほうが計算ノードよりコアあたりの性能が高いため，ビルド処理の速度を考慮したり，インタラクティブ環境の実行時間制限を考慮したりすると，このようなスーパーコンピュータにおいても，TLDT は威力を発揮できると考えられる．

これ以降，本稿は，以下の様な構成となっている．まず，2 章では，提案システム TLDT の設計と実装について述べる．続いて，3 章で，自動チューニング機能を有するソフトウェアを含むいくつかのソフトウェアを例に上げ，TLDT の便利さ，簡単さを紹介する．その後， 4 章で関連研究について述べた後，5 章でまとめを行う．

2. 設計と実装

我々は，TLDT (Transparent Load Distribution Tool) という，クロスコンパイルされた実行ファイルを，コンパイル環境から透過的に実行できるツールを開発した．本章では，このツールの設計と実装について述べる．

2.1 設計の概要

まず，プログラムが実行される場所には，「**ローカルホスト**」と「**リモートホスト**」の 2 種類があると仮定する．ローカルホストは，クロスコンパイラがインストールされており，プログラムがコンパイルされるコンパイル環境，例えば，スーパーコンピュータのログインノードを指す．リモートホストは，実際の意味のある計算が行われる実行環境，例えば，スーパーコンピュータの計算ノードを指す．また，実行可能ファイルにも，それぞれに対応する，「**ローカル実行ファイル**」と「**リモート実行ファイル**」の 2 種類があると仮定する．すなわち，ローカル実行ファイルはローカルホストで実行可能な実行ファイル，リモート実行ファイルは，例えばクロスコンパイラでコンパイルされたプログラムのように，リモートホストで実行可能な実行ファイルを指す．

透過的に両方の実行ファイルをローカルホストから実行できるようにするためには，**表 1** に示すような複数の場合を考慮する必要がある．ローカル実行ファイルをローカルホストで実行することは OS に実装されている通常の実行であるから，特に何も気にする必要はない．例えばローカルホストのアーキテクチャが x86_64 で，リモートホストのアーキテクチャが SPARC V9 であれば，ELF 形式の実行ファイルならファイルのヘッダを調べることでリモート実行ファイルか否かを判定することができる．このような場合，実行ファイルの実際の実行は `qsub`，`rsh` など，計

ⓒ 2012 Information Processing Society of Japan

表 1 実行の転送のために必要な機能一覧
Table 1 Function matrix necessary for execution forwarding

実行の場所	実行ファイルの形式	
	ローカル または 不明	リモート
ローカル	通常実行	代替実行ファイル
リモート	FORCE_LIST の使用	自動転送

算環境によって提供されるリモート実行用のコマンドを使用して，リモートホストで行うことができる．この，実行ファイルの形式の自動判定と，リモート実行ファイルである場合に実行を自動的にリモートホストで行うことを，“自動転送” と呼ぶ．この自動転送が TLDT の基本的な機能である．

実行ファイルが本来実行されるべき場所と，実行を行う場所が異なる場合は，透過的な実行を行うための仕掛け必要となる．例えば，シェルスクリプトの場合など，ローカル実行ファイルかリモート実行ファイルかを判定できないような場合，そのようなファイルをリモートホストで実行させたい場合，あるいは，ローカル実行ファイルが存在するのと同一のパスでリモートホストにも存在する実行ファイルを実行したい場合などは，TLDT にそのことを伝える必要がある．このためには，**FORCE_LIST** と呼ばれる実行ファイル一覧を作成する必要がある．このリストに記載された実行ファイルは，ファイルの形式にかかわらず，リモートホストでの実行を行うことにする．逆に，リモート実行ファイルをローカルホストで実行しようとする場合，当然そのまま実行することはできないが，同じ機能を持つローカル実行ファイルが別な場所に存在するとわかっている場合は，そのことを事前に TLDT に伝えてある場合は，リモート実行を行う代わりに，代わりの実行ファイルをローカルホストで実行できることにする．以下，それぞれの機能の詳細を述べる．

2.2 自動転送

自動転送を行うためには，実行ファイルの形式を自動的に判定することが重要である．自動判定ルーチンの現在の実装は，ELF [12] 形式の実行ファイルのヘッダ中の，`e_machine` フィールドの値から，その実行ファイルのアーキテクチャを調べている．

実行ファイルの形式がリモートである場合，リモートホストでプロセスマネージャを起動するための，実行環境に固有のシェルスクリプトを実行する．バッチジョブスケジューラで管理されている計算環境の場合，このスクリプトがバッチジョブを投入する．ローカルホストの TLDT は，実行ファイルのパス，引数文字列の配列 (ARGV) や，環境変数の定義 (ENVP) といった，`execve` システムコールの引数をプロセスマネージャに引き渡す．標準入力の中身についても同様にプロセスマネージャに渡される．リモートホストのプロセスマネージャは，これらの情報に基づいて実行を行い，標準出力，標準エラー出力，終了ステータスを返す．ローカルホストの TLDT と，リモートホストのプロセスマネージャの間の通信は，共有ファイルシステムのファイルまたは TCP ソケットを通して行われる．

2.3 FORCE_LIST

特定の実行ファイルを，形式にかかわらず，リモートホストで実行させたい場合は，コロン区切りのパスのリストを FORCE_LIST 環境変数に設定する．Grid Engine の，qtcsh [10] のような既存システムとは違い，基本的にはこの環境変数を設定しなくとも，ELF 形式の実行ファイルであれば自動転送が行われる．この機能は，例えば `uname` コマンドや `hostname` コマンドなど，ローカルホストとリモートホストで同一のパスに実行ファイルが存在するが，異なる結果を返すような場合に使用することが想定されている．また，シェルスクリプトなど，ELF 形式でない実行ファイルをリモートホストで実行させたい場合に利用することが可能である．

2.4 代替実行ファイル

現在の TLDT の実装では，自動転送が行われるたびにバッチジョブが投入される．バッチジョブの投入・実行には，たとえリソースが十分にある場合でもジョブ管理システムのポーリング間隔等に起因するオーバーヘッドがある．このため，ソフトウェアをビルドする際の，自動転送の回数を減らすことが出来れば，全体のビルド時間を短縮できる可能性がある．もし，ローカルホストに，リモートホストで実行する予定の実行ファイルと同じ振る舞いをする実行ファイルを準備できるのであれば，これを代替実行ファイルとして登録することで，TLDT はリモートホストへの自動転送を行う代わりに，代替実行ファイルをローカルホストで実行する．この機能は，ソースコードの依存性チェック等，アーキテクチャに依存しないような軽量の静的解析を行うプログラムに対して適用することが想定されている．実行ファイル **xxxxx** に対して，同一のディレクトリに **xxxxx.*altbin*** というローカル実行ファイルを置くことによって代替実行ファイルを登録する．

2.5 システムコールのフック

TLDT の実装は，`execve` システムコールの呼び出しをフックすることで実現されている．TLDT の機能は，代替の `execve` システムコールの実装を動的リンク可能ライブラリの形式で提供されている．Linux システムで，このライブラリを `LD_PRELOAD` 環境変数に設定することで利用可能である．現在のところ，TLDT には，FORCE_LIST，代替実行ファイルの処理を除く，基本的な “自動転送” 機能の

ⓒ 2012 Information Processing Society of Japan

表 2 Oakleaf-FX 計算ノードの仕様
Table 2 Oakleaf-FX Node Specifications

理論ピーク性能	236.5 GFLOPS / ノード
	14.8 GFLOPS / コア
CPU	SPARC64 IXfx 1.848 GHz
CPU 数 (コア数)	1 (16)
メモリ	32 GB
メモリバンド幅	85 GB / sec
ネットワーク	5 GB/s × 2 (双方向)
	× 4 (同時使用可能リンク数)

みが実装されている．TLDT を有効にした状態で execve が呼ばれると，実行ファイルの形式がリモートであるかがチェックされ，実行場所がリモートホストである場合，TLDT はジョブ投入を行いプロセスマネージャを起動し，プログラムの引数の転送・終了待ちを行う．

3. 事例研究

TLDT を利用することによって，クロスコンパイル環境においてはビルド時にリモートノードで得られる情報を必要とするような様々な既存のソフトウェアを，無変更，あるいは僅かな変更でビルドすることができる様になる．本章では，自動チューニング機能を有するソフトウェアを含む，このようなソフトウェアを TLDT を使用してビルドする例を紹介し，TLDT の利便性について述べる．

3.1 実験環境

実験は，東京大学情報基盤センターが提供する Oakleaf-FX システムの上で実施した．Oakleaf-FX システムは，富士通 PRIMEHPC FX10 スーパーコンピュータ [5] を 4,800 ノード使用して構成されている．CPU には， SPARC64 IXfx プロセッサが使用されている．SPARC64 IXfx プロセッサは 16 コア， 12 MB の 共有 L2 キャッシュを搭載し，周波数 1.848 GHz で動作する．各ノードのピーク性能は 236.5 GFLOPS で，チップあたり 115 W の電力を消費する．この CPU は SPARC v9 インストラクションセットアーキテクチャを，整数・浮動小数点演算のためのレジスタ数増加など，高性能計算用に拡張したものである．計算ノードの OS は Linux カーネル 2.6.25 を元に富士通がカスタマイズしたものである．計算ノード同士は，Tofu ネットワーク [1] と呼ばれる，6 次元メッシュ/トーラス型のネットワークで接続されており，一方向あたり最大 20 GB/秒の性能で通信を行うことが可能である．**表 2** に，Oakleaf-FX システムの計算ノードの仕様をまとめた．

Oakleaf-FX システムは，6 台のログインノードと，システムを対話的に利用するための 50 台のインタラクティブノードを提供している．各ログインノードは x86_64 アーキテクチャのサーバで，6 コアの Intel Xeon L5640 CPU (2.27 GHz) を 2 台搭載し，Hyper-Threading テクノロジを有効にして運用している．メモリ容量は 48 GB である．OS は RHEL6 (Linux カーネルのバージョンは 2.6.32) であり，富士通製クロスコンパイラが提供されている．インタラクティブノードは計算ノードと同一のアーキテクチャで，クロスコンパイラを利用したビルドができないようなソフトウェアをコンパイル・インストールしたり，並列プログラムをテスト・デバッグしたりする用途のために提供されている．1 ノード利用の場合は 2 時間，複数ノード利用の場合は 10 分の時間制限を設けて運用している．

3.2 ジョブ投入のオーバーヘッド

実験環境でのバッチジョブ投入のオーバーヘッドを調査するため，実行時間がほとんどかからないプログラムを TLDT 経由で実行した．具体的には，main 関数の中で何もせずに終了するプログラムを作成し，10 回連続でこれを実行して平均値を取り，1 回あたりの実行時間を求めた．その結果，1 回の実行あたり，平均して 1.70 秒かかることがわかった．

3.3 ATLAS

ATLAS (Automatically Tuned Linear Algebra Software) [3], [13] は，効率的な BLAS [4] 実装，そして LAPACK[2] の幾つかのルーチンへの C と Fortran77 のインタフェースを有する，数値計算ライブラリである．ATLAS はインストール先のプラットフォームに対して最適化されたライブラリを生成する，経験的なチューニング機構を備えている．これは実行環境で，様々な設定やカーネルルーチンを使用してベンチマーク計算を実施し，最適だったものを選択するという，典型的なインストール時自動チューニング機構である．

Oakleaf-FX システムは，計算ノード及びインタラクティブノードで実行可能な，「オウンコンパイラ」を提供している．我々は，計算ノードで，オウンコンパイラを使用し，バッチジョブとして ATLAS をビルドする場合と，ログインノードで TLDT を使用してビルドする場合の比較を行った．使用した ATLAS のバージョンは 3.8.4 である．`configure` 時の設定パラメータは，**表 3** に示すとおりである．どちらの場合も，アーキテクチャ等の自動検出が機能しなかったため，プラットフォームには 34 (Unknown UltraSPARC) を，オペレーティングシステムには 1 (Linux) を，アセンブラのタイプには 3 (SPARC) を，CPU 数には 16 を，そして CPU のクロック周波数には 1,848 MHz を手動で指定した．表の中で，`fcc` と `frt` は，オウンコンパイラ，`fccpx` と `frtpx` は，クロスコンパイラである．

ATLAS はビルド時に複数の種類のコンパイラを使用する．オウンコンパイラを使用する場合は，`fcc` と `frt`，そ

ⓒ 2012 Information Processing Society of Japan

表 3 ATLAS の Configure オプション
Table 3 Configure options on building ATLAS

オウンコンパイラ	-A 34 -O 1 -t 16 -s 3 -v 3 -m 1848 –cc=gcc
	-C acg ***fcc*** -F acg '-Xg -std=gnu99 -Kfast'
	-C xc gcc -F xc '-O'
	-C if ***frt*** -F if '-Kfast'
TLDT +	-A 34 -O 1 -t 16 -s 3 -v 3 -m 1848 –cc=gcc
クロスコンパイラ	-C acg ***fccpx*** -F acg '-Xg -std=gnu99 -Kfast'
	-C xc gcc -F xc '-O'
	-C if ***frtpx*** -F if '-Kfast'

表 4 ATLAS のビルド性能
Table 4 Performance summary of building ATLAS

オウンコンパイラ	
ビルド時間	20,880 秒

TLDT + クロスコンパイラ	
ビルド時間	13,260 秒
ジョブ投入回数	1,846

して，オウンコンパイラとして提供されている gcc のいずれかを対応させることでビルドが可能であるが，TLDT を使用する場合は，“GOODGCC” と呼ばれる種類のコンパイラをクロスコンパイラ (fccpx) に置き換える必要がある．しかし，“GOODGCC” の値は “gcc” にハードコードされていて，configure 時に変更することができない．このため，configure の実行後，37 個のディレクトリに生成された “Make.inc” に対して，“GOODGCC = gcc” という記述を “GOODGCC = fccpx” に手作業で置換する必要があった．TLDT を使用した場合，この小さな変更のみで，Oakleaf-FX の様なクロスコンパイル環境で ATLAS のビルド・インストールを行うことができた．

ATLAS のビルド性能をまとめたものを**表 4** に示す．オウンコンパイラでビルドした時と比べると，TLDT を使用してビルドした時の時間は 36%以上短縮されている事が分かる．この主な理由は，クロスコンパイラが実行されるログインノードが計算ノードよりも高速でメモリ容量も大きいためである．Oakleaf-FX システムでは，クロスコンパイラ環境でビルドできないソフトウェアをビルドするために，インタラクティブノードが提供されている．しかし，ATLAS のビルドには，インタラクティブノードの使用時間制限である 7,200 秒を超える時間がかかるため，TLDT を使用しなければ対話的環境で ATLAS のビルドを行うことはできなかった．

自動転送が行われた回数は， 1,846 回であった．ジョブ投入のオーバーヘッドに着目すると，この回数に 1.7 秒を掛けた 3,138 秒が，ジョブ投入のための待ち時間ということになる．このオーバーヘッドを取り除くことは今後の課題である．

3.4 ppOpen-AT

ppOpen-AT は，JST-CREST の研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」の研究課題である「自動チューニング機構を有するアプリケーション開発・実行環境」の中で開発されている，ppOpen-HPC [9] の主要なコンポーネントの一つである．

ppOpen-AT は ABCLibScript [8] を拡張したもので，現時点のバージョンでは，効率的なスレッド実行やレジスタあふれの削減に効果がある，ループ分割などの “Explicit methods” 向けの自動チューニング機能を提供している．ppOpon-AT はインストール時，実行時チューニングの双方の機能を有しているが，今回は，TLDT を使用して，そのインストール時チューニング機能を試した．実験の結果，以下の 3 つのサンプルプログラム (今回試したもののすべて) で，もともと ppOpen-AT のディレクティブが挿入されたソースコードに手を加えることなく，最適化を行うことができることを確認した．

- 行列-行列積
- 姫野ベンチマーク [6]
- BEM (境界要素法) コード

4. 関連研究

Grid Engine の，qtcsh [10] は，“透過的なジョブの分配” と呼ばれる機能を提供している．qtcsh は tcsh の拡張で，.qtask という設定ファイルのサポートを追加している．.qtask は，リモートホストで実行して欲しいコマンドの名前を定義し，Grid Engine における rsh 的なジョブ投入インタフェースである qrsh に渡すコマンドラインオプションも同時に定義することが出来る．qtcsh では，.qtask に記載されたコマンドが実行されようとすると，実際には qrsh が指定されたオプションで実行され，ジョブが投入される．ジョブの出力は qtcsh の出力にリダイレクトされる．その結果，利用者は qrsh の存在を意識することなく，コマンドは透過的に，適当な低負荷の Grid Engine 配下の計算ノードに転送される．

コマンド名の一致はシェル内で行われる．したがって，qtcsh がサブシェルを起動した場合，それがまた qtcsh でなければそのサブシェルの中に記述されたコマンドは，たとえ.qtask のリストに含まれるものであったとしてもリモートホストで実行されることはない．これらの点を考慮すると，qtcsh は，TLDT の FORCE_LIST 機能の一部を実装しているにすぎないと考えることができる．一方，我々のアプローチはシステムコールをフックするため，あらゆるシェルと一緒に利用することが可能である．さら

ⓒ 2012 Information Processing Society of Japan

に，実行ファイルを解析することにより自動的にリモートホストで実行する，自動転送機能のおかげで，設定ファイルを作成する労力を大幅に削減することができる．

HPC 分野では，システムコールのリモートホストへの転送は，I/O のために行われることがある．例えば， ZOID [7] 計算ノードと I/O ノード間のスケーラブルなアーキテクチャを提案している．また，清水らは，x86_64 アーキテクチャのノードと Cell/B.E. アーキテクチャのノードのヘテロジニアスクラスタで，実行と I/O の両方を転送するシステム [11] を提案している．このシステムではヘテロ性に対応したバイナリローダを Linux カーネルに統合して，複数の種類の実行ファイルの透過的な実行を可能にしている．一方，我々のアプローチでは，LD_PRELOAD 環境変数の仕組みを使用して，カーネルを変更せず，ライブラリを提供するだけで透過性を実現している．既存のスーパーコンピュータシステムは，利用者にシステムへの特権アクセスを許可しないポリシーで運用されているものが多いと考えられるので，カーネルの変更を行うことは大変困難である．この問題を考慮し，より多くのシステムでの利用を可能にするために，我々はカーネルを変更しないアプローチを採用している．

5. まとめと今後の課題

TLDT が有する実行透過性の効果で，ATLAS などの複数のクロスコンパイル対応していないソフトウェアを，無変更か，またはとても少ない変更でビルドすることができることが示された．FORCE_LIST や代替実行ファイルの機能などを実装することによってより多くのソフトウェアに TLDT を適用できるようになると考えられる．

また， 3.3 で述べたとおり，ジョブ投入のオーバーヘッドを減らすことも今後予定している．現在の実装では，自動転送が行われるたびにジョブが投入される．この結果，実装はとてもシンプルになるが，その一方で性能を犠牲にしている側面がある．我々は適当なタイムフレームの中の複数の自動転送イベントが，ひとつのプロセスマネージャ (ひとつのジョブに対応) を共有することで，ジョブ投入のオーバーヘッドを減らすことができるのではないかと考えている．

謝辞

本研究は科研費 MEXT/JSPS 基盤研究 (A) 「汎用自動チューニング機構を実現するためのソフトウェア基盤の研究」(23240005) の助成を受けたものである．また，ppOpen-AT についての実験では，東京大学情報基盤センターの片桐孝洋氏より 2012 年 4 月に更新された最新版のソースコードを提供して頂いた．

参考文献

[1] Ajima, Y., Sumimoto, S. and Shimizu, T.: Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers, *Computer*, Vol. 42, pp. 36–40 (2009).
[2] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D.: *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition (1999).
[3] Demmel, J., Dongarra, J., Eijkhout, V., Fuentes, E., Petitet, A., Vuduc, R., Whaley, R. C. and Yelick, K.: Self adapting linear algebra algorithms and software, *Proceedings of the IEEE*, p. 2005 (2005).
[4] Dongarra, J. J., Du Croz, J., Hammarling, S. and Duff, I. S.: A Set of Level 3 Basic Linear Algebra Subprograms, *ACM Trans. Math. Softw.*, Vol. 16, No. 1, pp. 1–17 (1990).
[5] Fujitsu: PRIMEHPC FX10 Supercomputer.
[6] Himeno, R.: Himeno benchmark.
[7] Iskra, K., Romein, J. W., Yoshii, K. and Beckman, P.: ZOID: I/O-forwarding infrastructure for petascale architectures, *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, PPoPP '08, New York, NY, USA, ACM, pp. 153–162 (2008).
[8] Katagiri, T., Kise, K., Honda, H. and Yuba, T.: ABCLibScript: a directive to support specification of an auto-tuning facility for numerical software, *Parallel Computing*, Vol. 32, No. 1, pp. 92 – 112 (2006).
[9] Nakajima, K., Satoh, M., Furumura, T., Okuda, H., Iwashita, T. and Sakaguchi, H.: ppOpen-HPC: Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications with Automatic Tuning, *IPSJ SIG Notes*, Vol. 2011, No. 44, pp. 1–9 (2011-07-20).
[10] Oracle Corporation: qtcsh in Oracle Grid Engine 6.2 User's Guide.
[11] Shimizu, M. and Yonezawa, A.: Remote Process Execution and Remote File I/O for Heterogeneous Processors in Cluster Systems, *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, Washington, DC, USA, IEEE Computer Society, pp. 145–154 (2010).
[12] TIS Committee: Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification, *Proceedings of the 12th ACM conference on Electronic commerce EC 11*, No. May, p. 29 (1995).
[13] Whaley, C., Petitet, A. and Dongarra, J. J.: Automated Empirical Optimization of Software and the ATLAS Project, *PARALLEL COMPUTING*, Vol. 27, p. 2001 (2000).

ⓒ 2012 Information Processing Society of Japan